Spring 2024: CS5720 Neural Networks & Deep Learning - ICP-7

Sai Deva Pranay Kumar Rao (700745063)

Image Classification with CNN

GitHub Link: https://github.com/raopranay1999/Neural-ICP7

Use Case Description:
LeNet5, AlexNet, Vgg16, Vgg19
1. Training the model
2. Evaluating the model

Programming elements:
1. About CNN
2. Hyperparameters of CNN
3. Image classification with CNN

In class programming:
1. Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy and reduce validation loss.
2. Provide logical description of which steps lead to improved response and what was its impact on architecture behavior.
3. Create at least two more visualizations using matplotlib (Other than provided in the source file)
4. Use dataset of your own choice and implement baseline models provided.
5. Apply modified architecture to your own selected dataset and train it.
6. Evaluate your model on testing set.
7. Save the improved model and use it for prediction on testing data
8. Provide plot of confusion matric
9. Provide Training and testing Loss and accuracy plots in one plot using subplot command and history object.
10. Provide at least two more visualizations reflecting your solution.
11. Provide logical description of which steps lead to improved response for new dataset when compared with baseline model and enhance architecture and what was its impact on architecture behavior.

kvn

Lenet

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import RMSprop, Adam
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

```
[3] (x_train, y_train), (x_test, y_test)  = keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 6s 0us/step
```

```
[4] classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

```
[5] y_train = y_train.reshape(-1,)
```

```
[6] # Reshape converting 2D to 1D
    y_test = y_test.reshape(-1,)
    y_train = y_train.reshape(-1,)
```

\+ Code + Text

```
[7] # This code normalazation
    x_train = x_train / 255.0
    x_test = x_test / 255.0
```

```
[8] x_train.shape
```

```
(50000, 32, 32, 3)
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

lenet = models.Sequential([
    layers.Conv2D(6, kernel_size=5, strides=1, activation='relu', input_shape=(32,32,3), padding='same'), #C1
    layers.AveragePooling2D(pool_size=(2, 2)), #S1
    layers.Conv2D(16, kernel_size=5, strides=1, activation='relu', padding='valid'), #C2
    layers.AveragePooling2D(pool_size=(2, 2)), #S2
    layers.Conv2D(120, kernel_size=5, strides=1, activation='relu', padding='valid'), #C3
    layers.Flatten(), #Flatten
    layers.Dense(84, activation='relu'), #F1
    layers.Dense(10, activation='softmax') #Output layer
])
```

```
[10] lenet.summary()
```

```
Model: "sequential"
```

```
lenet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         456

 average_pooling2d (Average  (None, 16, 16, 6)         0
 Pooling2D)

 conv2d_1 (Conv2D)           (None, 12, 12, 16)        2416

 average_pooling2d_1 (Avera  (None, 6, 6, 16)          0
 gePooling2D)

 conv2d_2 (Conv2D)           (None, 2, 2, 120)         48120

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 84)                40404

 dense_1 (Dense)             (None, 10)                850

=================================================================
Total params: 92246 (360.34 KB)
Trainable params: 92246 (360.34 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[11] lenet.compile(optimizer='adam', loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
```

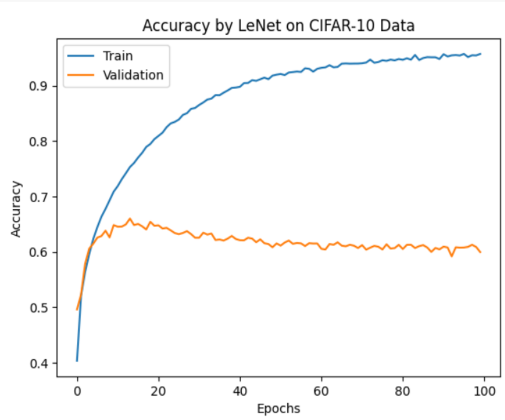
```
hist = lenet.fit(x_train, y_train, epochs=100, validation_data=(x_test, y_test),verbose=1)
```

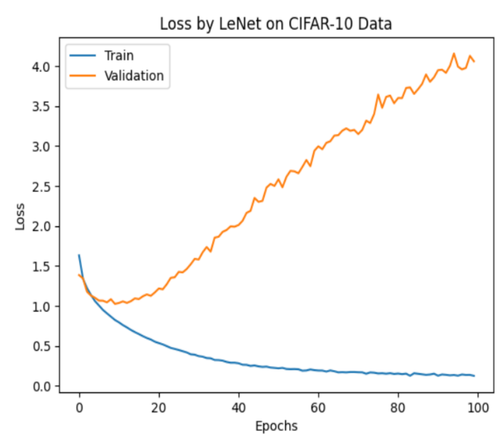
```
Epoch 1/100
1563/1563 [==============================] - 14s 6ms/step - loss: 1.6307 - accuracy: 0.4065 - val_loss: 1.4647 - val_accuracy: 0.4649
Epoch 2/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3603 - accuracy: 0.5140 - val_loss: 1.2848 - val_accuracy: 0.5421
Epoch 3/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.2405 - accuracy: 0.5598 - val_loss: 1.2675 - val_accuracy: 0.5519
Epoch 4/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1542 - accuracy: 0.5924 - val_loss: 1.1868 - val_accuracy: 0.5858
Epoch 5/100
1563/1563 [==============================] - 9s 6ms/step - loss: 1.0837 - accuracy: 0.6156 - val_loss: 1.1538 - val_accuracy: 0.5881
Epoch 6/100
1563/1563 [==============================] - 8s 5ms/step - loss: 1.0257 - accuracy: 0.6361 - val_loss: 1.1125 - val_accuracy: 0.6074
Epoch 7/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9690 - accuracy: 0.6582 - val_loss: 1.0944 - val_accuracy: 0.6150
Epoch 8/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.9188 - accuracy: 0.6763 - val_loss: 1.0767 - val_accuracy: 0.6275
Epoch 9/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8749 - accuracy: 0.6889 - val_loss: 1.0877 - val_accuracy: 0.6280
Epoch 10/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8358 - accuracy: 0.7040 - val_loss: 1.0876 - val_accuracy: 0.6247
Epoch 11/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.7975 - accuracy: 0.7184 - val_loss: 1.0880 - val_accuracy: 0.6293
Epoch 12/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.7582 - accuracy: 0.7331 - val_loss: 1.1409 - val_accuracy: 0.6155
Epoch 13/100
```

```
[ ] import numpy as np
```

```
[ ] # fix random seed for reproducibility
seed = 7
np.random.seed(seed)
```

```
# summarize history for accuracy
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title("Accuracy by LeNet on CIFAR-10 Data")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss by LeNet on CIFAR-10 Data')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'])
plt.show()
```

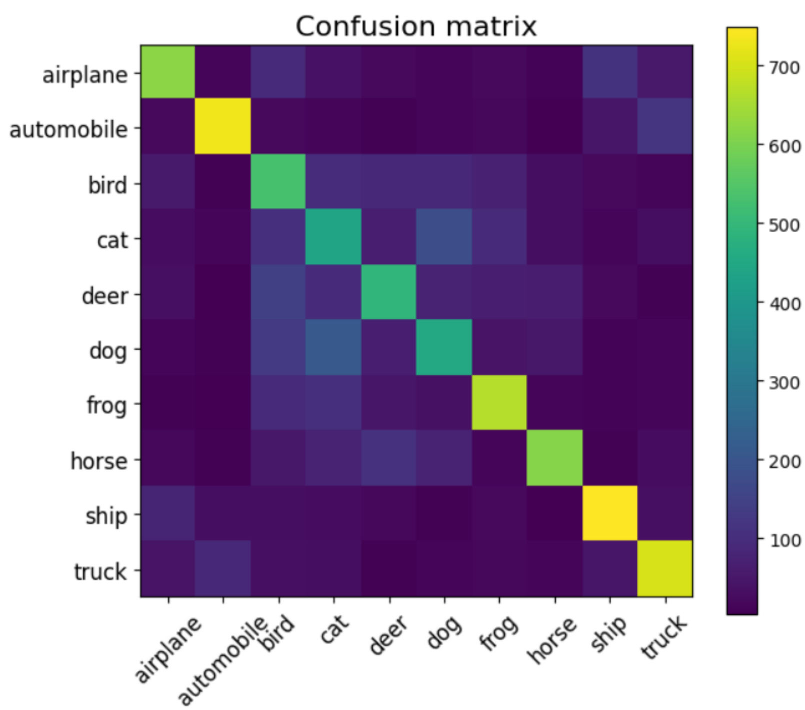




```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
y_predictions= lenet.predict(x_test)
y_predictions.reshape(-1,)
y_predictions= np.argmax(y_predictions, axis=1)

confusion_matrix(y_test, y_predictions)
```

```
313/313 [==============================] - 1s 2ms/step
array([[619,  15,  91,  40,  22,  14,  21,  11, 113,  54],
       [ 23, 731,  21,  16,   6,  12,  19,   5,  48, 119],
       [ 57,   6, 531,  97,  86,  87,  71,  30,  21,  14],
       [ 27,  12, 106, 439,  65, 179,  92,  32,  17,  31],
       [ 33,   3, 143,  95, 493,  73,  66,  64,  22,   8],
       [ 13,   8, 131, 211,  67, 453,  41,  50,  11,  15],
       [  8,   3,  92, 106,  48,  36, 666,  17,   9,  15],
       [ 19,   6,  50,  73, 110,  78,  17, 613,   6,  28],
       [ 80,  30,  30,  27,  19,   6,  21,   4, 748,  35],
       [ 43,  90,  34,  31,   7,  14,  18,  14,  47, 702]])
```

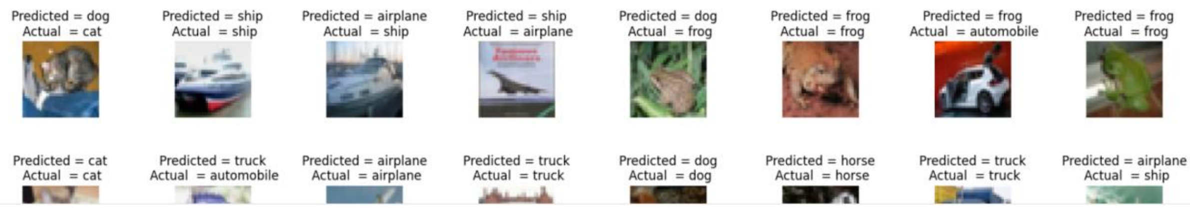
```
# confusion matrix and accuracy
from sklearn.metrics import confusion_matrix, accuracy_score
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions))
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```



```
[ ] print("Test accuracy:", accuracy_score(y_test, y_predictions))
```

```
Test accuracy: 0.5995
```

```
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```



AlexNet

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten
    from tensorflow.keras.optimizers import SGD, Adam
    from tensorflow.keras.layers import Convolution2D as Conv2D
    from tensorflow.keras.layers import MaxPooling2D
```

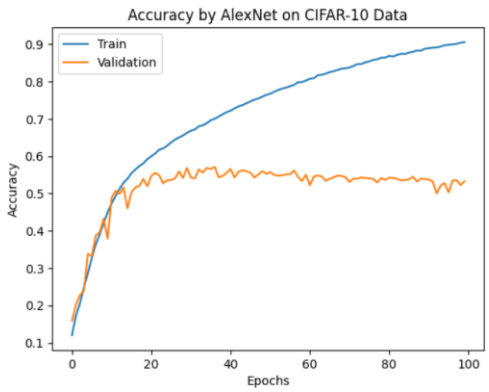
```
#Define Alexnet Model
AlexNet = Sequential()
AlexNet.add(Conv2D(filters=16,kernel_size=(3,3),strides=(4,4),input_shape=(32,32,3), activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(5,5),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(30,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(20,(3,3),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Flatten())
AlexNet.add(Dense(200, activation='relu'))
AlexNet.add(Dropout(0.1))
AlexNet.add(Dense(200, activation='relu'))
AlexNet.add(Dropout(0.1))
AlexNet.add(Dense(10,activation='softmax'))

AlexNet.compile(optimizer='SGD', loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
AlexNet.summary()
```

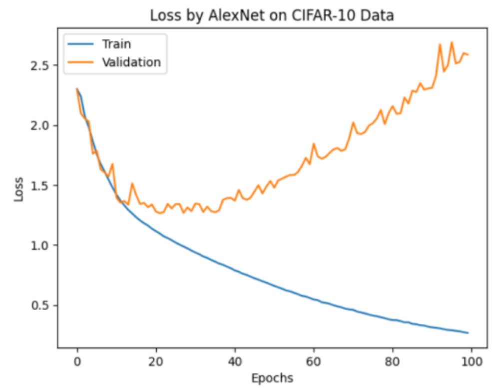
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 8, 8, 16)          448

 max_pooling2d (MaxPooling2  (None, 4, 4, 16)          0
 D)

 conv2d_4 (Conv2D)           (None, 4, 4, 60)          24060

 max_pooling2d_1 (MaxPoolin  (None, 2, 2, 60)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 2, 2, 60)          32460

 conv2d_6 (Conv2D)           (None, 2, 2, 30)          16230

 conv2d_7 (Conv2D)           (None, 2, 2, 20)          5420

 max_pooling2d_2 (MaxPoolin  (None, 1, 1, 20)          0
 g2D)

 flatten_1 (Flatten)         (None, 20)                0

 dense_2 (Dense)             (None, 200)               4200

 dropout (Dropout)           (None, 200)               0

  dense_3 (Dense)            (None, 200)              40200

  dropout_1 (Dropout)        (None, 200)              0

  dense_4 (Dense)            (None, 10)               2010

 =================================================================
 Total params: 125028 (488.39 KB)
 Trainable params: 125028 (488.39 KB)
 Non-trainable params: 0 (0.00 Byte)
```

```
history1 = AlexNet.fit(x_train, y_train, epochs=100, validation_data=(x_test, y_test),verbose=1)
```

```
Epoch 72/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4453 - accuracy: 0.8419 - val_loss: 1.9327 - val_accuracy: 0.5396
Epoch 73/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4386 - accuracy: 0.8468 - val_loss: 1.9244 - val_accuracy: 0.5395
Epoch 74/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4299 - accuracy: 0.8471 - val_loss: 1.9421 - val_accuracy: 0.5431
Epoch 75/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4207 - accuracy: 0.8517 - val_loss: 1.9950 - val_accuracy: 0.5415
Epoch 76/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4122 - accuracy: 0.8541 - val_loss: 2.0157 - val_accuracy: 0.5408
Epoch 77/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4066 - accuracy: 0.8580 - val_loss: 2.0550 - val_accuracy: 0.5386
Epoch 78/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3986 - accuracy: 0.8599 - val_loss: 2.1255 - val_accuracy: 0.5295
Epoch 79/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3907 - accuracy: 0.8638 - val_loss: 2.0074 - val_accuracy: 0.5413
Epoch 80/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3809 - accuracy: 0.8644 - val_loss: 2.0996 - val_accuracy: 0.5366
Epoch 81/100
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3752 - accuracy: 0.8688 - val_loss: 2.1578 - val_accuracy: 0.5424
Epoch 82/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3739 - accuracy: 0.8672 - val_loss: 2.0935 - val_accuracy: 0.5416
Epoch 83/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3657 - accuracy: 0.8713 - val_loss: 2.0978 - val_accuracy: 0.5394
```

```
# summarize history for accuracy
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title("Accuracy by AlexNet on CIFAR-10 Data")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss by AlexNet on CIFAR-10 Data')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'])
plt.show()
```

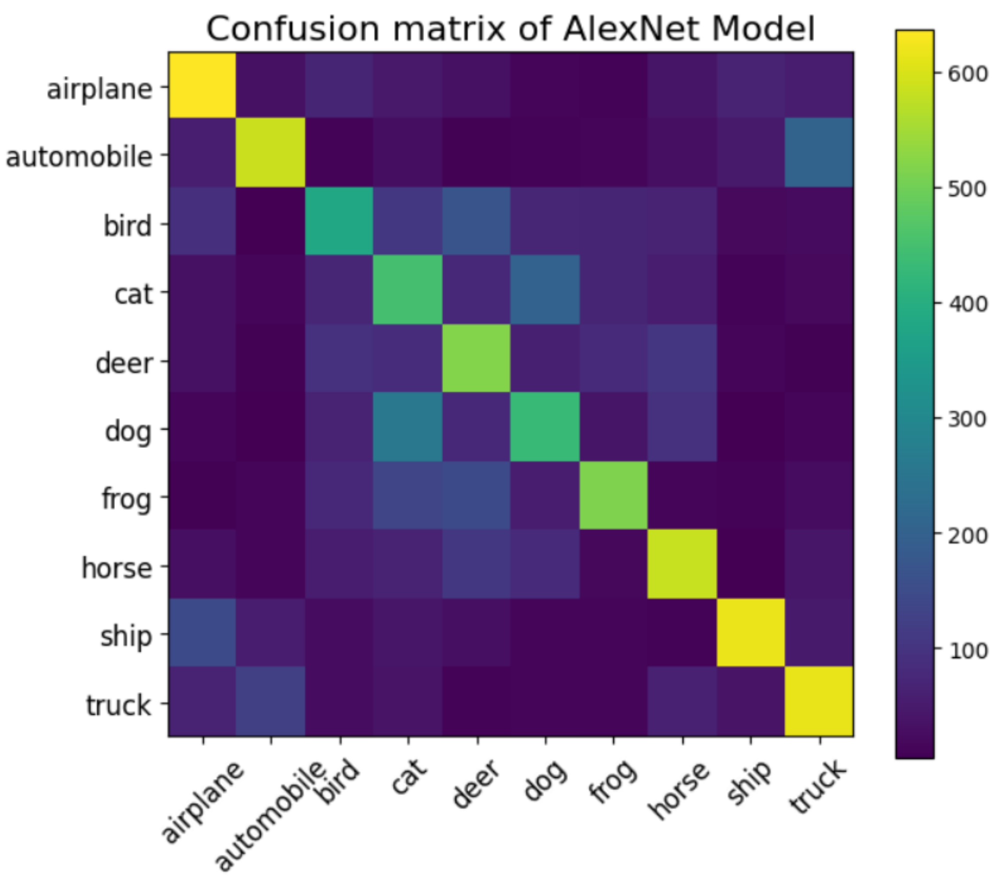





```
[ ]  y_predictions1 = AlexNet.predict(x_test)
     y_predictions1.reshape(-1,)
     y_predictions1= np.argmax(y_predictions1, axis=1)

     confusion_matrix(y_test, y_predictions1)
```

```
313/313 [==============================] - 1s 2ms/step
array([[636,  32,  69,  48,  32,  13,   9,  41,  66,  54],
       [ 57, 585,  10,  28,   8,  11,  15,  31,  50, 205],
       [ 90,   4, 381, 106, 169,  71,  70,  66,  21,  22],
       [ 33,  16,  73, 449,  75, 202,  70,  53,  10,  19],
       [ 32,   8,  93,  83, 519,  59,  82, 105,  12,   7],
       [ 14,   6,  64, 257,  75, 429,  40,  95,   4,  16],
       [  7,  12,  78, 136, 146,  53, 515,  16,  11,  26],
       [ 31,  14,  54,  65, 107,  82,  17, 582,   6,  42],
       [147,  52,  26,  42,  30,  15,  13,   9, 617,  49],
       [ 65, 124,  24,  37,   9,  16,  12,  62,  37, 614]])
```

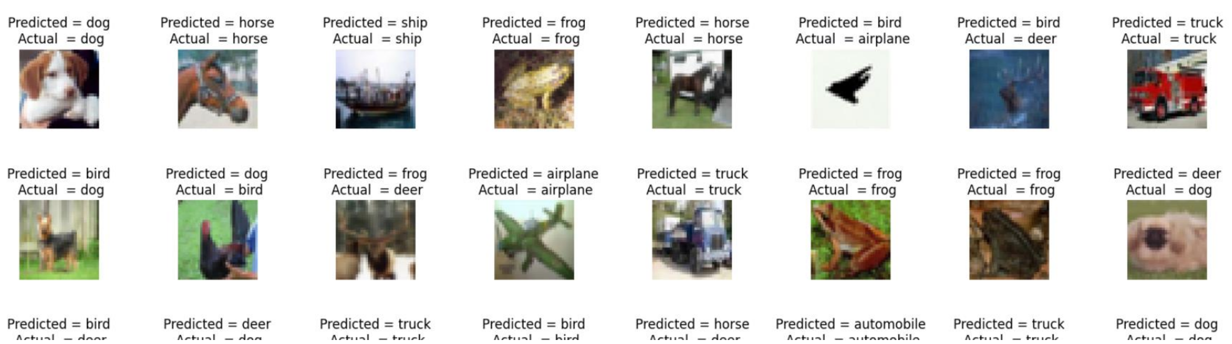
```python
# confusion matrix and accuracy
plt.figure(figsize=(7, 6))
plt.title('Confusion matrix of AlexNet Model', fontsize=16)
plt.imshow(confusion_matrix(y_test, y_predictions1))
plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), classes, fontsize=12)
plt.colorbar()
plt.show()
```




```python
[ ] print("Test accuracy by AlexNet:", accuracy_score(y_test, y_predictions))
```

```
Test accuracy by AlexNet: 0.5995
```

```python
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```



Vgg16

```
import keras
from keras.models import Sequential
from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D
from keras.datasets import cifar10
from keras import optimizers
from matplotlib import pyplot as plt
```

```
# generate cifar10 data
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

```
# config parameters
num_classes = 10
input_shape = x_train.shape[1:4]
optimizer = optimizers.Adam(lr=0.0003)
```

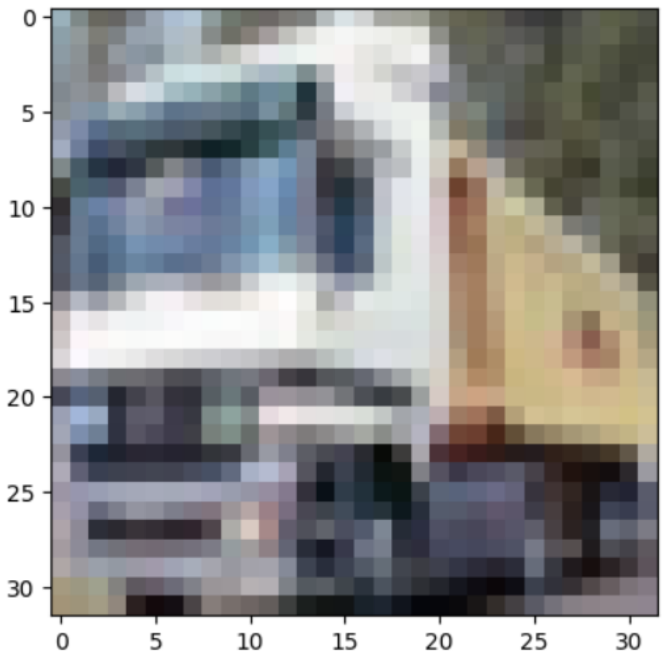
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# convert label to one-hot
one_hot_y_train = keras.utils.to_categorical(y_train,num_classes=num_classes)
one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

```
# check data
plt.imshow(x_train[1])
print(x_train[1].shape)
```

```
# check data
plt.imshow(x_train[1])
print(x_train[1].shape)
```

(32, 32, 3)



```
[ ] # build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(4096,activation='relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
# config optimizer,loss,metrics
model.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
```

```python
# check model
model.summary()
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 64)        1792

 conv2d_9 (Conv2D)           (None, 32, 32, 64)        36928

 max_pooling2d_3 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_11 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_4 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_12 (Conv2D)          (None, 8, 8, 256)         295168

 conv2d_13 (Conv2D)          (None, 8, 8, 256)         590080

 conv2d_14 (Conv2D)          (None, 8, 8, 256)         590080

 max_pooling2d_5 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 conv2d_15 (Conv2D)          (None, 4, 4, 512)         1180160

 conv2d_16 (Conv2D)          (None, 4, 4, 512)         2359808

 conv2d_17 (Conv2D)          (None, 4, 4, 512)         2359808

 max_pooling2d_6 (MaxPoolin  (None, 2, 2, 512)         0
 g2D)

 conv2d_18 (Conv2D)          (None, 2, 2, 512)         2359808

 conv2d_19 (Conv2D)          (None, 2, 2, 512)         2359808

 conv2d_20 (Conv2D)          (None, 2, 2, 512)         2359808

 max_pooling2d_7 (MaxPoolin  (None, 1, 1, 512)         0
 g2D)

 flatten_2 (Flatten)         (None, 512)               0

 dense_5 (Dense)             (None, 4096)              2101248

 dense_6 (Dense)             (None, 4096)              16781312

 dense_7 (Dense)             (None, 10)                40970

 activation (Activation)     (None, 10)                0

=================================================================
Total params: 33638218 (128.32 MB)
Trainable params: 33638218 (128.32 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# train
model.fit(x=x_train,y=one_hot_y_train,batch_size=128,epochs=10)
```

```
Epoch 1/10
391/391 [==============================] - 36s 68ms/step - loss: 2.3356 - accuracy: 0.0983
Epoch 2/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0959
Epoch 3/10
391/391 [==============================] - 23s 58ms/step - loss: 2.3027 - accuracy: 0.0961
Epoch 4/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0975
Epoch 5/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0969
Epoch 6/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0957
Epoch 7/10
391/391 [==============================] - 23s 58ms/step - loss: 2.3027 - accuracy: 0.0967
Epoch 8/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0959
Epoch 9/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0982
Epoch 10/10
391/391 [==============================] - 23s 59ms/step - loss: 2.3027 - accuracy: 0.0981
<keras.src.callbacks.History at 0x7b47ad77feb0>
```

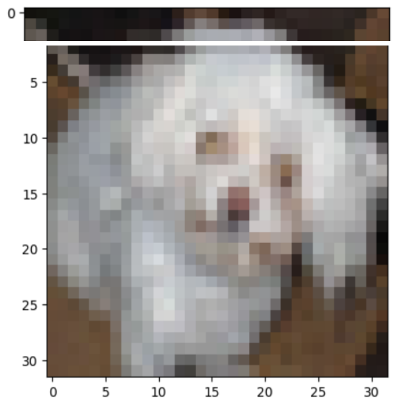
```python
# evaluate
print(model.metrics_names)
model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
# evaluate
print(model.metrics_names)
model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 7s 162ms/step - loss: 2.3026 - accuracy: 0.1000
[2.3025965690612793, 0.10000000149011612]
```

```
# predict
plt.imshow(x_test[1000])

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
expect = y_test[1000][0]
for i,_ in enumerate(result[0]):
    if result[0][i] > result[0][predict]:
        predict = i
print("predict class:",predict)
print("expected class:",expect)
```

```
1/1 [==============================] - 1s 640ms/step
predict class: 9
expected class: 5
```




```
] # save model
  model.save("keras-VGG16-cifar10.h5")
```

Vgg19

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import tensorflow as tf

    from tensorflow.keras.optimizers import RMSprop
    from keras.preprocessing import image
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization

    %matplotlib inline
```

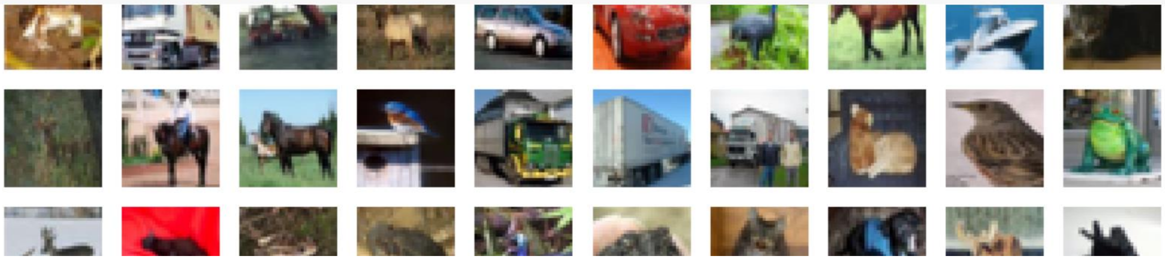
Extract data and train and test dataset

```
[ ] cifar100 = tf.keras.datasets.cifar100
    (X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()
```

```
[ ] classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

let's look into the dataset images

```
plt.figure(figsize = (16,16))
for i in range(100):
  plt.subplot(10,10,1+i)
  plt.axis('off')
  plt.imshow(X_train[i], cmap = 'gray')
```



Training, Testing and splitting data

```
[ ] from sklearn.model_selection import train_test_split
    x_train, x_val, y_train, y_val = train_test_split(X_train,Y_train,test_size=0.2)
```

```
[ ] from keras.utils import to_categorical
    y_train = to_categorical(y_train, num_classes = 10)
    y_val = to_categorical(y_val, num_classes = 10)
```

```
[ ] print(x_train.shape)
    print(y_train.shape)
    print(x_val.shape)
    print(y_val.shape)
    print(X_test.shape)
    print(Y_test.shape)
```

```
(40000, 32, 32, 3)
(40000, 10)
(10000, 32, 32, 3)
(10000, 10)
(10000, 32, 32, 3)
(10000, 1)
```

```
[ ] train_datagen = ImageDataGenerator(
        preprocessing_function = tf.keras.applications.vgg19.preprocess_input,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range = 0.1,
        height_shift_range = 0.1,
        shear_range = 0.1,
        horizontal_flip = True
    )
    train_datagen.fit(x_train)

    val_datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.vgg19.preprocess_input)
    val_datagen.fit(x_val)
```

```
[ ] from keras.callbacks import ReduceLROnPlateau
    learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                                patience=3,
                                                verbose=1,
                                                factor=0.5,
                                                min_lr=0.00001)
```

```python
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights="imagenet",
    input_shape=(32,32,3),
)

vgg_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0         
                                                                 
 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0         
                                                                 
 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080    
```

```python
model = tf.keras.Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384  
                                                                 
 flatten_3 (Flatten)         (None, 512)               0         
                                                                 
 dense_8 (Dense)             (None, 1024)              525312    
                                                                 
 dense_9 (Dense)             (None, 1024)              1049600   
                                                                 
 dense_10 (Dense)            (None, 256)               262400    
                                                                 
 dense_11 (Dense)            (None, 10)                2570      
                                                                 
=================================================================
Total params: 21864266 (83.41 MB)
Trainable params: 21864266 (83.41 MB)
Non-trainable params: 0 (0.00 Byte)
```

```python
optimizer = tf.keras.optimizers.SGD(lr = 0.001, momentum = 0.9)
model.compile(optimizer= optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

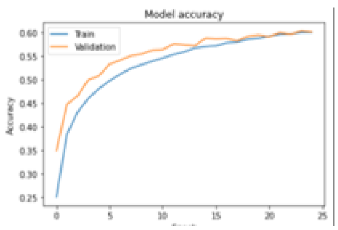
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
```

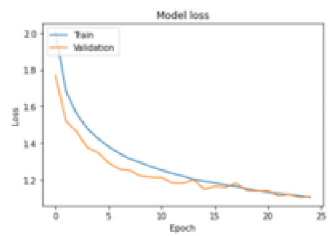
```python
history = model.fit(
    train_datagen.flow(x_train, y_train, batch_size = 128),
    validation_data = val_datagen.flow(x_val,y_val, batch_size = 128),
    epochs = 25,
    verbose = 1,
    callbacks = [learning_rate_reduction]
)
```

```
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
313/313 [==============================] - 34s 109ms/step - loss: nan - accuracy: 0.0998 - val_loss: nan - val_accuracy: 0.1009 - lr: 0.0100
Epoch 5/25
313/313 [==============================] - 33s 105ms/step - loss: nan - accuracy: 0.0998 - val_loss: nan - val_accuracy: 0.1009 - lr: 0.0050
Epoch 6/25
313/313 [==============================] - 31s 100ms/step - loss: nan - accuracy: 0.0998 - val_loss: nan - val_accuracy: 0.1009 - lr: 0.0050
Epoch 7/25
313/313 [==============================] - ETA: 0s - loss: nan - accuracy: 0.0998
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0024999999441206455.
313/313 [==============================] - 32s 103ms/step - loss: nan - accuracy: 0.0998 - val_loss: nan - val_accuracy: 0.1009 - lr: 0.0050
```

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure()
plt.plot(acc,color = 'purple',label = 'Training Acuracy')
plt.plot(val_acc,color = 'blue',label = 'Validation Accuracy')
plt.legend()
```

```python
acc = history.history['loss']
val_acc = history.history['val_loss']

plt.figure()
plt.plot(acc,color = 'purple',label = 'Training Loss')
plt.plot(val_acc,color = 'blue',label = 'Validation Loss')
plt.legend()
```





Using the code above, the CIFAR-10 dataset—a well-known collection of photographs divided into 10 classes—is loaded. The data is used to construct testing and training sets. The input images are converted from integers to floats and normalized between 0 and 1. The output labels are encoded using a one-hot process.

Next, the dimensions of the input data are changed to match the expected input shape of the convolutional neural network (CNN).

The CNN model is then defined using the Keras Sequential API. It consists of many convolutional layers with ReLU activation, dropout layers to prevent overfitting, and dense layers with ReLU activation. Sort the pictures into the ten groups. Sort the pictures into the ten groups. Put the picture into each of the ten classes. The optimizer in the model is stochastic gradient descent (SGD), the loss function is categorical cross-entropy, and the performance measure for the training phase is accuracy.For each of the 25 epochs of model training, 32 batches are employed. model's accuracy is printed once it has been assessed on the test set.

Next, the model is used to forecast the images in the test set. The projected labels and the actual labels are compared to see if the model has predicted correctly. Finally, the accuracy and loss are demonstrated using the history object that the fit() method returned.